Final Report: Combining Interprocedural Compile-Time and Run-Time Parallelization

DARPA contract DABT63-95-C-0118

USC/ISI Principal Investigator: Mary Hall
Caltech Principal Investigator: Carl Kesselman

This report describes the final results of a project revolving around an integrated automatic parallelization system that combines high quality interprocedural compile-time analysis with flexible run-time support. This research builds on our previous extensive work on interprocedural compile-time analysis for parallelization, sponsored by DARPA through contracts at Rice University and Stanford University. The system is designed to address three core issues: (1) improving the effectiveness of compile-time parallelization analysis; (2) integration of compile-time analysis with efficient run-time parallelization tests; and (3) managing an efficient mapping of parallel constructs to a target architecture.

This project was begun while Mary Hall, the original Principal Investigator, was a visiting professor at Caltech. When Mary Hall moved to USC/ISI in June, 1996, the contracting agent requested that the contract be moved to ISI as a subcontract to Caltech. For this reason, a Caltech PI was named, Carl Kesselman, and Mary Hall was listed as a subcontractor.

This project has led to a large number of publications in prestigious conferences and journals, and the five most significant of these are attached to this report. This document summarizes these results; more detail can be found in the accompanying publications.

1 Key Results

The overall project consisted of four core research component areas, each of which will be described in this section.

1.1 Improved Compile-Time Analysis

This project began as a continuation to a several-year, DARPA-sponsored effort at Stanford University (DARPA Contract F30602-95-C-0098) in which the PI participated as a Stanford research associate prior to her positions at Caltech and University of Southern California. The goal of this large effort at Stanford was to develop an automatic parallelizing compiler as part of the SUIF compiler that surpassed the effectiveness of any other commercial or research system of the time.

The Stanford effort led to the most extensive experimental results on automatic parallelization ever disclosed, including a recommendation and breakdown of the analyses that must be added to other systems to replicate these results.

As the current project began, the results of this Stanford effort were still being gathered. As the first step in the current project, we worked in collaboration with the group at Stanford headed by Monica Lam to extend the automatic parallelization system as needed to parallelize the SPECFP95 BENCHMARKS, to complement previous results on the SPECFP92, NAS and PERFECT benchmark programs. We also documented all of these results, and they have been summarized in two important papers that were written during the current contract (see references [1] and [2]).

The most significant result appeared in reference [1]. We demonstrated through a combination of parallelization analysis and locality optimizations results on SPECFP95 benchmarks that were 50% better than any previously reported results. We also performed extensive experimentation across programs in four benchmark suites (see reference [2]). We found that the new analysis techniques found in the Stanford SUIF system, namely interprocedural array data-flow analysis, array privatization and array reduction recognition, were critical to parallelizing significant coarse-grain computations that could yield parallel speedups on modern multiprocessors. We found that a third of these programs spend more than 50% of their exeuction time in computations that are parallelized with these techniques. We also found that the interprocedural analysis enabled parallelizing very large loops, containing up to 1000 lines of code.

1.2 Instrumentation System for Identifying Remaining Parallelism

While the Stanford effort led to an overall successful automatic parallelization system, there were still some remaining programs that yielded little or no parallel speedup. These results raised the following questions: is the compiler exploiting all the available parallelism in these applications, and if not, what new analyses can be developed to exploit the remaining parallelism? To answer these questions, we built a system that instruments array and scalar memory accesses in loops left sequential by the base SUIF parallelizer. At run time, the instrumentation indicates which of these loops are actually parallelizable with the standard program input. Through this tool combined with our high-quality parallelization analysis, we performed the first empirical measurement of the gap between automatic parallelization and inherent parallelism in programs. This experiment was performed across programs in the three benchmark suites, SPECFP95, NAS and PERFECT (see reference [3]).

Our experiment found that the compiler was exploiting most of the available parallelism in these applications, but it identified two major areas in which current parallelizing compiler technology could be significantly improved.

- compiler analysis must take control flow paths into account to improve the precision of its result.
- many loops are parallelizable only under certain program inputs, so run-time tests are required to guarantee safety of parallelization.

system. With this modified data-flow analysis called *predicated data-flow analysis*, the compiler derives, in addition to the usual conservative data-flow results, optimistic results guarded by predicates that guarantee their correctness. These predicates can be used to derive conditions under which dependences can be eliminated or privatization is possible. These conditions can be used both to enhance compile-time analysis and to introduce run-time tests that guard safe execution of a parallelized version of a computation.

As compared to previous work that combines predicates with array data-flow analysis, our approach is distinguished by two features: (1) it derives low-cost, run-time parallelization tests; and, (2) it incorporates *predicate embedding* and *predicate extraction*, which translate between the domain of predicates and data-flow values to derive more precise analysis results. We present extensive experimental results across three benchmark suites and one additional program, demonstrating that predicated array data-flow analysis parallelizes more than 40% of the remaining inherently parallel loops left unparallelized by the SUIF compiler (as determined by the instrumentation experiment) and that it yields improved speedups for 5 programs (see reference [4]).

1.4 Adaptive Parallel Thread Allocation

We developed an approach to map parallel computations to a target architecture to improve utilization of computer resources (see attached reference [5]). Most compilers use an all-or-nothing static mapping of computations to processors: the computation executes on all the processors, or if it is deemed too fine grained, executes sequentially. This standard approach works fine on small numbers of processors such as the 8-processor target machines used in previous work on automatic parallelizing compilers. But on larger numbers of processors, this approach can lead to a significant waste of resources, where parallel computations executed on increasing numbers of processors yield diminishing returns. This waste of resources becomes more acute on parallel computers used as multiprogrammed compute engines, where processors not being effectively used could be applied to useful work in other applications.

To address this limitation, we developed and evaluated an extension to the SUIF run-time system that adapts the number of processors allocated to a computation at run time, based on how effectively it is using processor resources during the *current execution* of the program. This approach not only adapts to varying amounts of parallelism within a single application but also responds to increases in a multiprogrammed workload.

In collaboration with Margaret Martonosi at Princeton University, we performed on an experiment on five programs from the SPECFP95 and NAS benchmark suite, examining 2 and 3 program workloads on a 14-processor 95 MHz SGI Power Challenge system. We found that our run-time approach improved workload performance up to 33% over one-at-a-time runs of the workload.

2 Publications Describing this Work

This project has supported research that has led to numerous publications in some of the premier journals and conferences of parallel computing. We list the 11 most relevant publications in this section. The first five of these will be included with the final report.

Five Most Significant Publications. The first five publications summarize the work described in the previous section. They are numbered, and these numbers are used in references in the preceding descriptions.

[1] "Maximizing Multiprocessor Performance with the SUIF Compiler," M. Hall, J. Anderson, S. Amarasinghe, B. Murphy, E. Bugnion and M. Lam. *IEEE Computer* 29(12) (Dec. 1996).

[2] "Interprocedural Parallelization Analysis in SUIF," by M.W. Hall, S. Amarasinghe, B. Murphy, S. Liao and M. Lam. Manuscript accepted for publication in *ACM Transactions on Programming Languages and Systems*, to appear.

[3] "Evaluating Automatic Parallelization in SUIF," by B. So, S. Moon and M. W. Hall, Manuscript accepted for publication in *IEEE Transactions on Parallel and Distributed Systems*, to appear.

[4] "Evaluation of Predicated Array Data-Flow Analysis for Automatic Parallelization," by S. Moon and M.W. Hall, in *Proceedings of the ACM Symposium on Principles and Practice of Parallel Programming*, May, 1999.

[5] "Adaptive Parallelism in Compiler-Parallelized Code," by M.W. Hall and M. Martonosi, *Concurrency: Practice and Experience*, 10(14) (1998).

Other Journal Publications. A few additional journal publications related to this project are as follows.

"Combining Compile-Time and Run-Time Parallelization," S. Moon, B. So and M.W. Hall, Invited paper from LCR '98 in *Scientific Programming*, to appear.

"Multiprocessors from a Software Perspective," S. Amarasinghe, J. Anderson, C.S. Wilson, S. Liao, B. Murphy R. French, M.S. Lam, M.W. Hall. Invited paper selected as a Hot Chips '95 award paper, IEEE Micro 16(3) (Jun. 1996).

Other Conference and Workshop Publications. These are additional refereed papers appearing in workshops and conferences.

"Measuring the Effectiveness of Automatic Parallelization in SUIF," Byoungro So, Sungdo Moon and Mary Hall, to appear in International Conference on Supercomputing, Melbourne, Australia, July, 1998.

"Predicated Array Data-Flow Analysis for Run-Time Parallelization," Sungdo Moon, Mary Hall and Brian Murphy, to appear in International Conference on Supercomputing, Melbourne, Australia, July, 1998.

"A Case for Combining Compile-Time and Run-Time Parallelization," S. Moon, B. So, M.W. Hall, and B. Murphy, in *Proceedings of the Workshop on Languages, Compilers and Run-time Systems for Parallel Computing*, Pittsburgh, May, 1998.

"Detecting Coarse-Grain Parallelism Using an Interprocedural Parallelizing Compiler," M.W. Hall, S.P. Amarasinghe, B.R. Murphy, S. Liao and M.S. Lam, in *Proceedings of Supercomputing '95*, December '95.

3 Major Deliverable

In addition to the numerous publications describing the work, listed above, there was a major deliverable in the form of released software.

We released the compiler-parallelized Specfp95 applications that have been used in experiments described by publications that are part of this project. We arranged with the SPEC committee to release these applications to licensed SPEC users, and we have provided them with a copy of all the code and accompanying Makefiles and information. We envision these applications will be the compiler equivalent of the hand-parallelized SPLASH-2 benchmarks, and that they will become commonly used by members of the architecture research community in evaluating system features.

Information on this release can be found on the SPEC website,
http://www.specbench.org/osg/cpu95/par-research, and is also announced on the Stanford SUIF website http://suif.stanford.edu.

4 Personnel Involved

This project was primarily carried out by the Principal Investigator, Mary Hall, and her two USC Phd students, Sungdo Moon and Byoungro So. The adaptive thread management work in Section 1.4 was a collaborative effort with Margaret Martonosi at Princeton, and included work by research programmer Kimo Yap. The early work described in Section 1.1 involved collaboration with Monica Lam and her students at Stanford.

REPORTDOCUMENTATIONPAGE

FormApproved
OMBNo.07040188

1.REPORTDATE *(DDMMYYYY)* 05-05-1999	2 Report Type: Final Technical	3.DATESCOVERED *(FromTo)* 09-11-1995 - 09-10-1998

4.TITLEANDSUBTITLE Effective Parallelization Via Interprocedural Compile-Time, Run-Time and Interactive Techniques	5a.CONTRACTNUMBER DABT 63-95-C-0118 5b.GRANTNUMBER N/A 5c.PROGRAMELEMENTNUMBER
6.AUTHOR(S) Carl Kesselman Mary Hall	5d.PROJECTNUMBER 5e.TASKNUMBER 5f.WORKUNITNUMBER
7.PERFORMINGORGANIZATIONNAME(S)ANDADDRESS(ES) California Institute of Technology Mail Code 213-6 Pasadena, CA 91125	8.PERFORMINGORGANIZATION REPORTNUMBER
9.SPONSORING/MONITORINGAGENCYNAME(S)ANDADDRESS(ES) Directorate of Contracting Attn: A7ZS-DKO-I Post Office Box 12748 Ft. Huachuca, AZ 85670-2748	10.SPONSOR/MONITOR'SACRONYM(S) 11.SPONSORING/MONITORING AGENCYREPORTNUMBER

12.DISTRIBUTIONAVAILABILITYSTATEMENT

Approved for public release; distribution is unlimited

13.SUPPLEMENTARYNOTES

14.ABSTRACT

This report describes the final results of a project revolving around an integrated allelization system that combines high quality interprocedural compile-time analysis with flexible run-time support. This research builds on our previous extensive work on interprocedural compile-time analysis for parallelization, sponsored by DARPA through contracts at Rice University and Stanford University. The system is designed to address three core-issues: (1) improving the effectiveness of compile-time parallelization analysis; (2) integration of compile-time analysis with efficient run-time parallelization tests; and (3) managing an efficient mapping of parallel contracts to a target architecture

15.SUBJECTTERMS

Parallel Computing; Compilers; Run-time Systems

16.SECURITYCLASSIFICATIONOF:			17.LIMITATIONOF ABSTRACT	18.NUMBER OFPAGES	19a.NAMEOFRESPONSIBLEPERSON
a.REPORT	b.ABSTRACT	c.THISPAGE			19b.TELEPONENUMBER(*Includeareacode)*

StandardForm298(Rev.898)
PrescribedbyANSIStdZ3918

DTIC QUALITY INSPECTED 4